BIT 관련 알고리즘

- 비트 검색용 알고리즘을 분석 한다.
- 비트 관련 알고리즘을 분석한다.

1) 비트 검색 알고리즘
2) 비트 관련 알고리즘

ffs 알고리즘

find first set bit : O(n)

10000000 00000000 00000000 00000000

```
int my_ffs( int bitmap )
{
    int i;
    for( i=0; i<32; i++ )
        if( bitmap & (1<<i) )
            break;
    return i;
}
```

ffs 알고리즘

find first set bit : $O(\log_2 n)$

```
int num = 0;

if ((word & 0xffff) == 0) {
    num += 16;
    word >>= 16;
}
if ((word & 0xff) == 0) {
    num += 8;
    word >>= 8;
}
if ((word & 0xf) == 0) {
    num += 4;
    word >>= 4;
}
if ((word & 0x3) == 0) {
    num += 2;
    word >>= 2;
}
if ((word & 0x1) == 0)
    num += 1;
return num;
```

24 num

```
00000001 00000000 00000000 00000000
00000000 00000000 11111111 11111111 &
-------------------------------------
00000000 00000000 00000000 00000000

00000001 00000000
00000000 11111111 &
-------------------
00000000 00000000

00000001
00001111 &
---------
00000001
```

ffs 알고리즘

find first set bit : $O(\log_2 n)$

```
int num = 0;

if ((word & 0xffff) == 0) {
    num += 16;
    word >>= 16;
}
if ((word & 0xff) == 0) {
    num += 8;
    word >>= 8;
}
if ((word & 0xf) == 0) {
    num += 4;
    word >>= 4;
}
if ((word & 0x3) == 0) {
    num += 2;
    word >>= 2;
}
if ((word & 0x1) == 0)
    num += 1;
return num;
```

31 num

```
10000000 00000000 00000000 00000000
00000000 00000000 11111111 11111111 &
---------------------------------------
00000000 00000000 00000000 00000000

10000000 00000000
00000000 11111111 &
-------------------
00000000 00000000

10000000
00001111 &
---------
00000000

1000
0011 &
---------
0000

10
01 &
---------
00
```

ffs 알고리즘

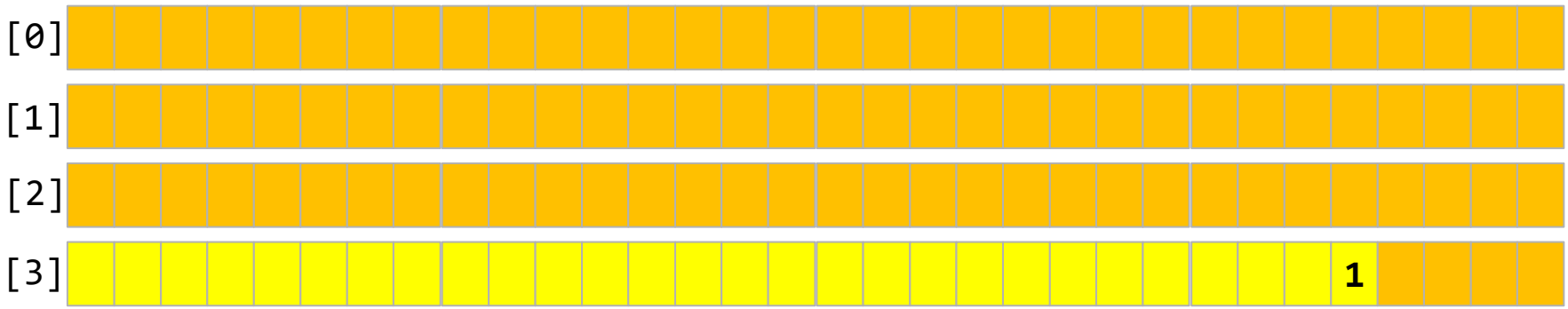
find first set bit : $O(\log_2 n)$

```
int num = 0;

if ((word & 0xffff) == 0) {
    num += 16;
    word >>= 16;
}
if ((word & 0xff) == 0) {
    num += 8;
    word >>= 8;
}
if ((word & 0xf) == 0) {
    num += 4;
    word >>= 4;
}
if ((word & 0x3) == 0) {
    num += 2;
    word >>= 2;
}
if ((word & 0x1) == 0)
    num += 1;
return num;
```

31 num

```
00000000 00000000 00000000 00000000
00000000 00000000 11111111 11111111 &
-------------------------------------
00000000 00000000 00000000 00000000

00000000 00000000
00000000 11111111 &
------------------
00000000 00000000

00000000
00001111 &
---------
00000000

0000
0011 &
---------
0000

00
01 &
---------
00
```

find_next_bit 알고리즘

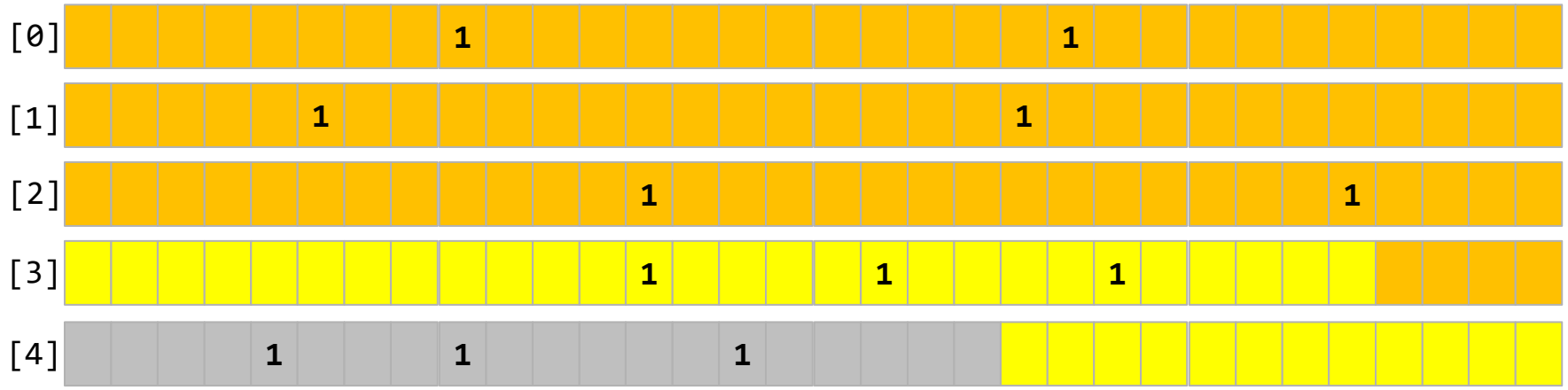


100-32 => 68-32 => 36-32 => 4
몫:3 , 나머지:4

```
int bitmap[4] = {0,};
bitmap[3] |= 1 << 4;
bitmap[100/32] |= 1<<(100%32);
```

find_next_bit 알고리즘

RT Process : 0 ~ 99
Normal Process : 100 ~ 139



100-32 => 68-32 => 36-32 => 4
몫:3 , 나머지:4

```
int bitmap[4] = {0,};
bitmap[3] |= 1 << 4;
bitmap[100/32] |= 1<<(100%32);
```

hamming weight 알고리즘

0x12345678 => O(n)

0001 0010 0011 0100 0101 0110 0111 1000

```
int bit_count( int bitmap )
{
    int count=0;
    int i;
    for( i=0; i<32; i++ )
       if( bitmap & (1<<i) )
           count++;
    return count;
}
```

hamming weight 알고리즘

```
const int m1  = 0x55555555;
const int m2  = 0x33333333;
const int m4  = 0x0f0f0f0f;
const int m8  = 0x00ff00ff;
const int m16 = 0x0000ffff;
const int h01 = 0x01010101;

int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

0x12345678 => O($\log_2 n$)

```
00010010001101000101011001111000

00010010001101000101011001111000
01010101010101010101010101010101 &
00010000000101000101010001010000

00001001000110100010101100111100
01010101010101010101010101010101 &
00000001000100000000000100010100

00010000000101000101010001010000
00000001000100000000000100010100 +
00010001001001000101010101100100
```

```
00 01 00 10 00 11 01 00 01 01 01 10 01 11 10 00
00 01 00 01 00 10 01 00 01 01 01 01 01 10 01 00
```

hamming weight 알고리즘

```
const int m1  = 0x55555555;
const int m2  = 0x33333333;
const int m4  = 0x0f0f0f0f;
const int m8  = 0x00ff00ff;
const int m16 = 0x0000ffff;
const int h01 = 0x01010101;

int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

0x12345678 => $O(\log_2 n)$

```
00010001001001000101010101100100

00010001001001000101010101100100
00110011001100110011001100110011 &
00010001001000000001000100100000

00000100010010010001010101011001
00110011001100110011001100110011 &
00000000000000010001000100010001

00010001001000000001000100100000
00000000000000010001000100010001 +
00010001001000010010001000110001
```

```
0001 0010 0011 0100 0101 0110 0111 1000
0001 0001 0010 0001 0010 0010 0011 0001
```

hamming weight 알고리즘

```
const int m1  = 0x55555555;
const int m2  = 0x33333333;
const int m4  = 0x0f0f0f0f;
const int m8  = 0x00ff00ff;
const int m16 = 0x0000ffff;
const int h01 = 0x01010101;

int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

0x12345678 => $O(\log_2 n)$

```
00010001001000010010001000110001

00010001001000010010001000110001
00001111000011110000111100001111 &
00000001000000010000000100000001

00000001000100010000100100010011
00001111000011110000111100001111 &
00000001000000010000000100000011

00000001000000010000000100000001
00000001000000010000000100000011 +
00000010000000110000010000000100
```

```
00010010 00110100 01010110 01111000
00000010 00000011 00000100 00000100
```

hamming weight 알고리즘

```
const int m1  = 0x55555555;
const int m2  = 0x33333333;
const int m4  = 0x0f0f0f0f;
const int m8  = 0x00ff00ff;
const int m16 = 0x0000ffff;
const int h01 = 0x01010101;

int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

0x12345678 => O($\log_2 n$)

```
00000010 00000011 00000100 00000100
00000000 11111111 00000000 11111111
00000000 00000011 00000000 00000100

00000000 00000010 00000011 00000100
00000000 11111111 00000000 11111111
00000000 00000010 00000000 00000100

00000000 00000011 00000000 00000100
00000000 00000010 00000000 00000100+
00000000 00000101 00000000 00001000

0000000000000000 0000000000001101
```

hamming weight 알고리즘

```
int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

```
int popcount_b(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    x += x >>  8;
    x += x >> 16;
    return x & 0x7f;
}
```

0x12345678 => $O(\log_2 n)$

```
00010010001101000101011001111000
01010101010101010101010101010101 &
00010000000101000101010001010000

00001001000110100010101100111100
01010101010101010101010101010101 &
00000001000100000000000100010100

00010000000101000101010001010000
00000001000100000000000100010100 +
00010001001001000101010101100100

00010010001101000101011001111000
00000001000100000000000100010100 -
00010001001001000101010101100100

00010001001001000101010101100100
00010001001001000101010101100100
```

hamming weight 알고리즘

```
int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

a	b	count
0	0	00
0	1	01
1	0	01
1	1	10

```
int popcount_b(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    x += x >>  8;
    x += x >> 16;
    return x & 0x7f;
}
```

a	b	count
0	0	00
0	1	01
1	0	01
1	1	10

```
00      01
 0 -     0 -
00      01

10      11
 1 -     1 -
01      10
```

hamming weight 알고리즘

```
int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

```
int popcount_b(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    x += x >>  8;
    x += x >> 16;
    return x & 0x7f;
}
```

```
0001 0001 0010 0001 0010 0010 0011 0001
0000 0010 0000 0011 0000 0100 0000 0100

0001 0001 0010 0001 0010 0010 0011 0001
0000 0001 0001 0010 0001 0010 0010 0011 +
0001 0010 0011 0011 0011 0100 0100 0100

0001 0010 0011 0011 0011 0100 0100 0100
0000 1111 0000 1111 0000 1111 0000 1111 &
0000 0010 0000 0011 0000 0100 0000 0100
0000 0010 0000 0011 0000 0100 0000 0100
```

hamming weight 알고리즘

```
int popcount_a(int x)
{
    x = (x & m1 ) + ((x >>  1) & m1 );
    x = (x & m2 ) + ((x >>  2) & m2 );
    x = (x & m4 ) + ((x >>  4) & m4 );
    x = (x & m8 ) + ((x >>  8) & m8 );
    x = (x & m16) + ((x >> 16) & m16);
    return x;
}
```

```
00000010 00000011 00000100 00000100
00000000 00000010 00000011 00000100 +
00000010 00000101 00000111 00001000
```

```
int popcount_b(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    x += x >>  8;
    x += x >> 16;
    return x & 0x7f;
}
```

```
0000001000000101 0000011100001000
                 0000001000000101
0000001000000101 0000100100001101

0000001000000101 0000100100001101
0000000000000000 0000000001111111 &
0000000000000000 0000000000001101
```

hamming weight 알고리즘

```
const int h01 = 0x01010101

int popcount_b(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    x += x >>  8;
    x += x >> 16;
    return x & 0x7f;
}

int popcount_c(int x)
{
    x -= (x >> 1) & m1;
    x = (x & m2) + ((x >> 2) & m2);
    x = (x + (x >> 4)) & m4;
    return (x * h01) >> 24;
}
```

```
                    0x02030404
                    0x01010101 *
-----------------------------
                    0x02030404
                  0x02030404
                0x02030404
              0x02030404        +
-----------------------------
              0x0205090d0b0804
                      0000000d
```

hamming weight 알고리즘

```
int popcount_d(int x)
{
    int count;
    for (count=0; x; count++)
        x &= x - 1;
    return count;
}
```

3 count

```
00010010001101000101011001111000
00010010001101000101011001110111 &
00010010001101000101011001110000
00010010001101000101011001101111 &
00010010001101000101011001100000
00010010001101000101011001011111 &
00010010001101000101011001000000
```

hamming weight 알고리즘

```
int popcount_d(int x)
{
    int count;
    for (count=0; x; count++)
        x &= x - 1;
    return count;
}
```

2	count

특성 : 세팅된 비트가 $\log_2 N$ 미만인 경우 유리한 알고리즘

```
00010000000010000000000000000000
00010000000001111111111111111111 &
00010000000000000000000000000000
00001111111111111111111111111111 &
00000000000000000000000000000000
```

bit reverse 알고리즘

```
char ch = 0x12; => 0x48
```

```
00010010 => 0x12
01001000 => 0x48
```

bit reverse 알고리즘

```
u16 ch = 0x1234; => 0x48
```

```
00010010 00110100 => 0x1234
00101100 01001000 => 0x2c48
```

bit reverse 알고리즘

```
u32 ch = 0x12345678; => 0x1e6a2c48
```

```
00010010 00110100 01010110 01111000 => 0x12345678
00011110 01101010 00101100 01001000 => 0x1e6a2c48
```